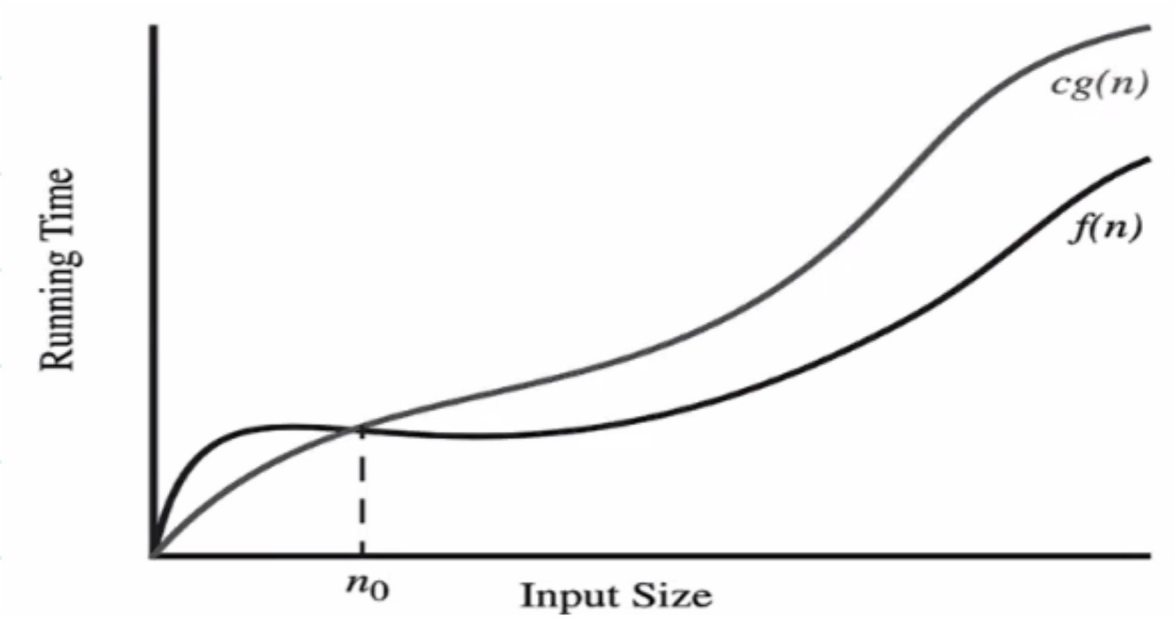
Big-Oh

Let $f(n)$ and $g(n)$ be functions mapping nonnegative integers to real numbers. We say that $f(n)$ is $O(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for every integer $n \geq n_0$. This definition is often pronounced as "$f(n)$ is ***big-Oh*** of $g(n)$" or "$f(n)$ is ***order*** $g(n)$." (See Figure 1.5.)



f(n) - How behavior of alg changes as n changes

cgn(n) - Always more complex than f(n) after a certain n

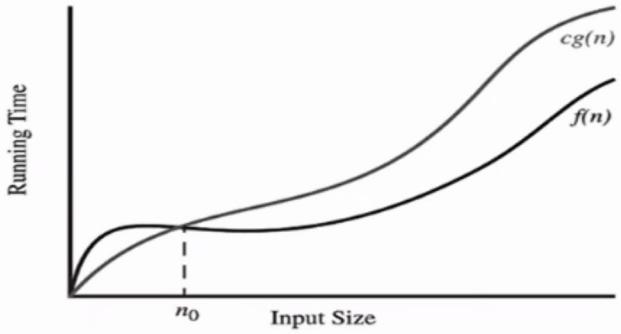
Prove factorial algorithm is $O(n)$

$f(n) = 4n + 3$

$c = 5$? $4n + 3 \leq 5n$

$n_0 = 3$

- Find a g(n) that is asymptotically small.



- Find a function that characterizes how the algorithm changes based on the input size

- Ignore constants (don't change based on input)

Big-Oh simplification

Theorem 1.7: *Let $d(n)$, $e(n)$, $f(n)$, and $g(n)$ be functions mapping nonnegative integers to nonnegative reals.*

1. *If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a > 0$.*
2. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)+e(n)$ is $O(f(n)+g(n))$.*
3. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.*
4. *If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.*
5. *If $f(n)$ is a polynomial of degree d (that is, $f(n) = a_0 + a_1 n + \cdots + a_d n^d$), then $f(n)$ is $O(n^d)$.*
6. *n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$.*
7. *$\log n^x$ is $O(\log n)$ for any fixed $x > 0$.*
8. *$\log^x n$ is $O(n^y)$ for any fixed constants $x > 0$ and $y > 0$.*

Prove that $2n^3 + 4n^2\log(n)$ is $O(n^3)$

∴ 2: $O(2n^3) + O(4n^2\log(n))$ is $O(2n^3 + 4n^2\log(n))$

∴ Case 8: $\log(n)$ is $O(n)$

∴ Case 1: $4n^2$ is $O(n^2)$

∴ Case 3: $4n^2\log n$ is $O(n^3)$

∴ Case 2: $2n^3 + 4n^2\log(n)$ is $O(n^3)$

Common functions

Some Functions Ordered by Growth Rate	Common Name
$\log n$	logarithmic
$\log^2 n$	polylogarithmic
$\sqrt{n}$	square root
n	linear
$n \log n$	linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential

"n log n"

Other Notations

Big Omega and Big Theta

Similar to Big-Oh

- Big Omega allows to say asymptotically that a function is greater than or equal to another up to a constant factor.

$\Omega(\ )$

- Big Theta allows to say that two functions are asymptotically equal up to a constant factor.

$\Theta(\ )$

Interpreting asymptotic notation

- Order the following functions

2^{100} 4^n $loglogn$ $log^2 n$ 2^{logn}

2^{100} - Not tied to n

loglogn - Slower than normal log n

$log^2 n$ -

$2^{logn} - 2^{logn} = n$

What algorithm should I prefer?

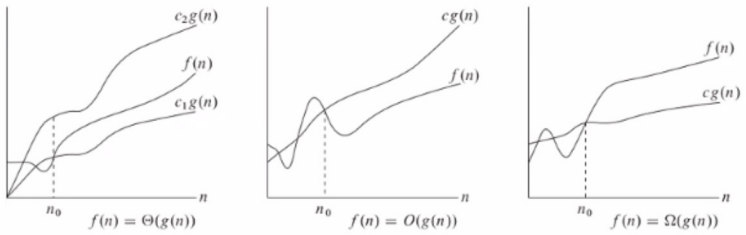
Theta(10^{100}n) or Theta(10nlogn)

- Theta(n) vs Theta(nlogn)

- Formally: Theta(n)
- Practically: Theta(nlogn)
  - 10^{100} is a googol times n, astronomically large

Exercise 2

- Suppose that an algorithm A for some problem takes time asymptotic to n^2 with one input I_0 of size n, where there are many other inputs of sizes n. Use *Big-X* (Θ, Ω, O) notations to answer the following questions:
  - *Q1. How would you describe A's best-case complexity?*
  - *Q2. How would you describe A's worst-case complexity?*
  - *Q3. How would you describe A's average-case complexity?*



Q1. We know of one input that results in n^2 , but there may be better ones. Suppose $f(n)$ is the best-case complexity of A, then $f(n) \leq n^2$ for any n, n^2 provides an upper bound of $f(n)$.
Big-O notation applies: $f(n) \leq cn^2$ and $f(n) = O(n^2)$.

Q2. In opposition to Q1, suppose $f(n)$ is the worst-case complexity of A. we know $f(n) \geq n^2$ for any n. n^2 provides a lower bound of $f(n)$.
Big-Omega notation applies: $f(n) \geq cn^2$ and $f(n) = \Omega(n^2)$.

Q3. It cannot be determined as we only have the information for one input. We need more information (probability distribution, etc.) of the inputs to answer the average-case complexity.

CLEMSON
School of COMPUTING

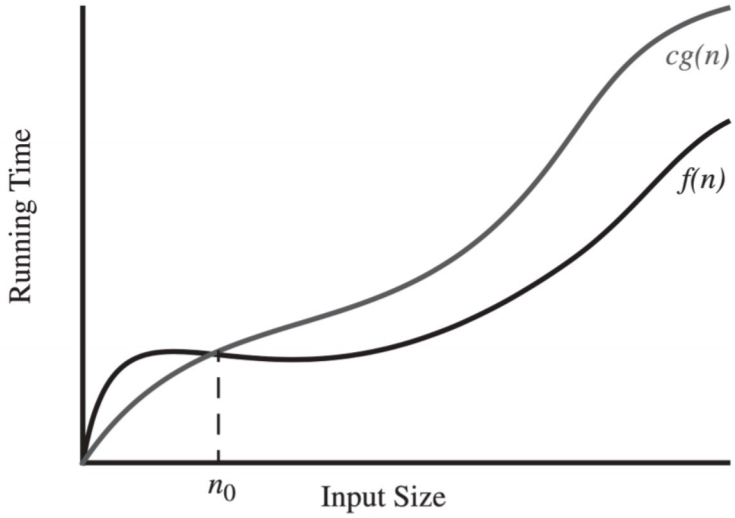
Review on Asymptotic notation

Why asymptotic notation?

- RAM model forces us to a laborious analysis of the algorithm's steps
- Most instructions do not depend on the input size and are less interesting to analyze
- The asymptotic notation allows us to focus on such operations only.

The Big-Oh notation

Let $f(n)$ and $g(n)$ be functions mapping nonnegative integers to real numbers. We say that $f(n)$ is $O(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for every integer $n \geq n_0$. This definition is often pronounced as "$f(n)$ is ***big-Oh*** of $g(n)$" or "$f(n)$ is ***order*** $g(n)$." (See Figure 1.5.)



The Big-Oh notation

Let $f(n)$ and $g(n)$ be functions mapping nonnegative integers to real numbers. We say that $f(n)$ is $O(g(n))$ if there is a real constant $c > 0$ and an integer constant $n_0 \geq 1$ such that $f(n) \leq cg(n)$ for every integer $n \geq n_0$. This definition is often pronounced as "$f(n)$ is ***big-Oh*** of $g(n)$" or "$f(n)$ is ***order*** $g(n)$." (See Figure 1.5.)

How to apply the Big-Oh definition?

- Prove that the factorial algorithm is O(n)

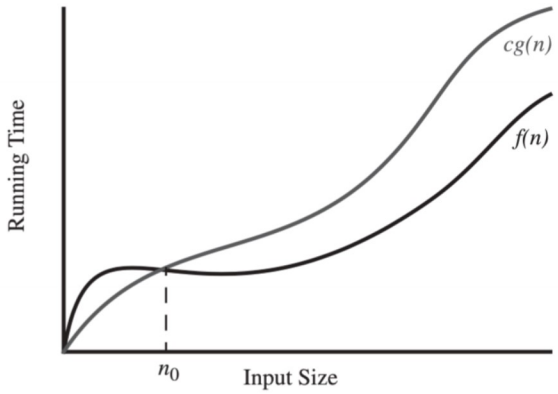
$f(n) = 4n + 3$

$c=5 \quad => \quad 4n+3 <= 5n$

$n_0=3$

How to use the Big-Oh notation

- Find a g(n) that is asymptotically small.



- Find a function that characterizes how your algorithm changes based on the input size

How to use the Big-Oh notation

Theorem 1.7: *Let $d(n)$, $e(n)$, $f(n)$, and $g(n)$ be functions mapping nonnegative integers to nonnegative reals.*

1. *If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a > 0$.*
2. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)+e(n)$ is $O(f(n)+g(n))$.*
3. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.*
4. *If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.*
5. *If $f(n)$ is a polynomial of degree d (that is, $f(n) = a_0 + a_1 n + \cdots + a_d n^d$), then $f(n)$ is $O(n^d)$.*
6. *n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$.*
7. *$\log n^x$ is $O(\log n)$ for any fixed $x > 0$.*
8. *$\log^x n$ is $O(n^y)$ for any fixed constants $x > 0$ and $y > 0$.*

How to use the Big-Oh notation

Prove that $2n^3 + 4n^2\log n$ is $O(n^3)$

Case 8 – $\log n$ is $O(n)$

Case 1 - $4n^2$ is $O(n^2)$

Case 3 - $4n^2\log n$ is $O(n^3)$

Case 1 – $2n^3$ is $O(n^3)$

Case 2 – $2n^3 + 4n^2\log n$ is $O(n^3)$

Theorem 1.7: *Let $d(n)$, $e(n)$, $f(n)$, and $g(n)$ be functions mapping nonnegative integers to nonnegative reals.*

1. *If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a > 0$.*
2. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)+e(n)$ is $O(f(n)+g(n))$.*
3. *If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.*
4. *If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.*
5. *If $f(n)$ is a polynomial of degree d (that is, $f(n) = a_0 + a_1 n + \cdots + a_d n^d$), then $f(n)$ is $O(n^d)$.*
6. *n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$.*
7. *$\log n^x$ is $O(\log n)$ for any fixed $x > 0$.*
8. *$\log^x n$ is $O(n^y)$ for any fixed constants $x > 0$ and $y > 0$.*

Common functions

Some Functions Ordered by Growth Rate	Common Name
$\log n$	logarithmic
$\log^2 n$	polylogarithmic
$\sqrt{n}$	square root
n	linear
$n \log n$	linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential

Big Omega and Big Theta

Similar to Big-Oh

- Big Omega allows to say asymptotically that a function is greater than or equal to another up to a constant factor.

- Big Theta allows to say that two functions are asymptotically equal up to a constant factor.

Interpreting asymptotic notation

- Order the following functions

2^{100} 4^n $loglogn$ $log^2 n$ 2^{logn}

2^{100}

$loglogn$

$log^2 n$

$2^{logn} = n$

4^n

Interpreting asymptotic notation

What algorithm should I prefer?

Theta(10^{100}n) or Theta(10nlogn)

- Formal point of view - Theta(n) vs Theta(nlogn)
- Practical point of view - Theta(n) vs Theta(nlogn)

What's next?

- Book chapters
  - Chapter 1.1
- In-class activities
  - Practice with Big-Oh notation
- Next class
  - Amortized analysis

Big-Oh Notation

Exercise 1

- Prove that the Fibonacci algorithm (for-loop version) is $O(n)$.

Exercise 1

- Prove that the Fibonacci algorithm (for-loop version) is $O(n)$.

Recall that $f(n) = 6n$, and thus we find $c = 6$ such that $f(n) \leq 6 \cdot n$ when $n > 0$.

Exercise 1

- Prove that the Fibonacci algorithm (for-loop version) is $O(n^2)$.

Exercise 1

- Prove that the Fibonacci algorithm (for-loop version) is $O(n^2)$.

$f(n) = 6n \leq c \cdot n^2$ (pick $c = 1$)

→ $6n \leq n^2$ (omit n from both sides)

→ $6 \leq n$ ($n_0 = 6$)

So that $f(n) \leq n^2$ when $n \geq 6$.

Exercise 1

- Which one should we pick for the Fibonacci algorithm? $O(n)$ or $O(n^2)$? Why?

Exercise 1

- Which one should we pick for the Fibonacci algorithm? $O(n)$ or $O(n^2)$? Why?

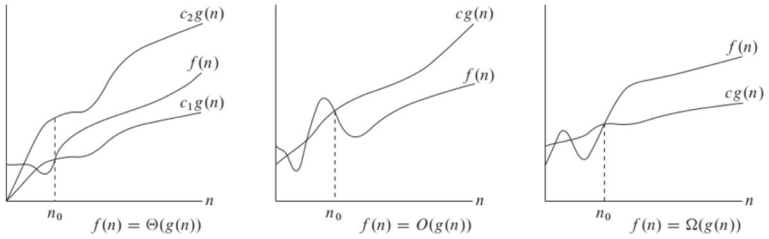
$O(n)$. Because Big-Oh defines the upper bound of an algorithm, and we want to limit the upper bound as small as possible.

Think about a set $S = \{5, 14, 23, 40\}$ and upper bounds 100000000 vs. 40

By using $O(n)$, we know that for the worst-case scenario, the Fibonacci algorithm takes linear time.

Exercise 2

- Suppose that an algorithm A for some problem takes time asymptotic to n^2 with one input I_0 of size n, where there are many other inputs of sizes n. Use *Big-X* (Θ, Ω, O) notations to answer the following questions:
  - *Q1. How would you describe A's best-case complexity?*
  - *Q2. How would you describe A's worst-case complexity?*
  - *Q3. How would you describe A's average-case complexity?*



Exercise 2

Q1. We know of one input that results in n^2 , but there may be better ones. Suppose $f(n)$ is the best-case complexity of A, then $f(n) \leq n^2$ for any n, n^2 provides an upper bound of $f(n)$.
Big-O notation applies: $f(n) \leq cn^2$ and $f(n) = O(n^2)$.

Q2. In opposition to Q1, suppose $f(n)$ is the worst-case complexity of A. we know $f(n) \geq n^2$ for any n. n^2 provides a lower bouncomplexityd of $f(n)$.
Big-Omega notation applies: $f(n) \geq cn^2$ and $f(n) = \Omega(n^2)$.

Q3. It cannot be determined as we only have the information for one input. We need more information (probability distribution, etc.) of the inputs to answer the average-case complexity.